AT&T Bell Laboratories

subject: **Improving Test Case Selection with Constrained Arrays**

date: **March 26, 1990**

from: **G. B. Sherwood**
MT XT9152400
2K339 x72743
attmail!gsherwood

TECHNICAL MEMORANDUM

1. Introduction

This memorandum describes a system test tool which analyzes possible test cases and suggests a sequence of tests to minimize the number of untested combinations of test factors. The goals are the same as that for the OATS[1] tool, namely

- **Savings in testing time** due to a reduced number of tests cases, and
- **Improved ability to find faults** because of better coverage of combinations of test factors.

However, the approach taken here is very different.

The Constrained Array Test System (CATS) takes as input sets of test factor values which span the test cases under consideration. These cases can be derived directly from an analysis of requirements; they can be taken from OATS output, etc. CATS reorders these cases using an algorithm that looks for the next "best" test case, which minimizes the number of untested combinations of factors. CATS output is the reordered test cases. With each test case is a running count of untested combinations of factor values. The user can judge how many of the test cases to execute, based on this information. CATS can analyze the combinations using an arbitrary number of factors at a time, limited only by the number of factors in the problem or the computing resources in use.

The CATS prototype is available without formal support to interested users.

2. Motivation

The motivation to prototype CATS came from the application of OATS to the Client installation test cases for AT&T StarGROUP™ Software V3.3. This particular test job was very large and and labor intensive: The requirements[2] called for 101 combinations of personal computers (PCs) and operating system versions to be used with fifteen different network cards. We wanted to see if OATS could reduce the number of test cases below the 41 cases we had previously selected, and if the coverage of combinations of factors could be improved.

In retrospect, the operating system version of the client PC was an important factor for finding software defects. However, this factor is dependent on two others -- the vendor of the PC and its processor. PC vendors specify which versions of DOS and OS/2 are supported on any particular model; generally new machines are not supported with earlier operating systems. OS/2 cannot be used with 8088 or 8086 processors, and only one of our required PC vendors supports all required processors.

Matters get more complex when network cards are considered. Our requirements call for support for two different PC bus types. The newer of these is supported by only one of our required PC vendors, and only on newer processors and newer operating system versions. I.e. the bus type is dependent on three other factors.

OATS provides for some dependencies among test factors. However, the real constraints contained in our requirements cannot be given to OATS without some simplification. We can *omit* some of the dependencies, but that leads to some of OATS' test cases being impossible. For example, anachronistic configurations like an 8088 processor with a Microchannel bus or an 80386 processor with DOS 3.1 are suggested. If these impossible test cases are left out, then some of the desired combinations of factors are not covered.

Thus the motivation for CATS was a simple way to account for complex constraints among the test factors. By taking as input all test cases under consideration, CATS adopts the constraints implicitly.

3. Findings

The following conclusions result from use of the CATS prototype.

3.1 Practicality

The process (described in Section 4) is practical. Large arrays can be processed in reasonable periods of time. Currently "large" means up to 250 rows or test cases, up to 64 factors, and up to 100,000 combinations of factor values. These limits can be redefined easily.

3.2 Detection of Higher Order Mode Failures

The process used by CATS is not restricted to work with factor values two at a time. It is easy to consider single, double, triple or higher mode failure detection if such faults are thought to be important to the test job. By default, CATS minimizes the untested combinations for all possible mode failures up to the smaller of 1) the number of factors in the problem, or 2) the practical limit based on computing resources.

3.3 Conformance to Test Case Constraints

Our need to accommodate complex constraints is satisfied by CATS, as illustrated in the following example. Each of the 101 required PC and operating system combinations was listed as a test case for analysis by CATS.[1] Four factors were used in this example:

1. PC vendor, with four values,
2. Processor, with five values,
3. PC bus type, with two values, and
4. Operating system version, with six values.

The output from CATS (illustrated in the following tables) is just reordered test cases from the input. So the constraints of the problem cannot be violated. For example, in the CATS OUTPUT -- PART 1 Table[2]

1. Detailed consideration of network cards and other relevant factors is beyond the scope of this paper.
2. The first column of each row labels the row; the next four columns correspond to the four factors; the next four columns list the number of untested combinations of factor values taken one, two, three or four at a time; and the last column totals the number of untested combinations. The row labeled 0 lists the number of values for each factor and the the numbers of untested combinations before any tests are executed. For example, there are 4 + 5 + 2 + 6 = 17 combinations of factor values taken one at a time; there are 4 * 5 * 2 * 6 = 240 combinations of factor values taken four at a time. Each subsequent row contains the factor values for its test case and the numbers of untested combinations when all cases through the current one are executed.

the row labeled 1 corresponds to the first configuration listed in our requirements, an IBM PC AT running PC DOS 3.1. The row labeled 16 corresponds to the sixteenth configuration in our requirements, a Compaq 386 - S running Compaq DOS 3.2, etc.

CATS OUTPUT -- PART 1									
0	4	5	2	6	1:17	2:104	3:268	4:240	629
1	ibm	80286	xt/at	dos3.1	1:13	2:98	3:264	4:239	614
16	compaq	80386	xt/at	dos3.2	1:10	2:92	3:260	4:238	600
19	ibm	80386sx	mca	dos3.3	1:7	2:86	3:256	4:237	586
28	at&t-o	8086	xt/at	dos4.0	1:4	2:80	3:252	4:236	572
100	at&t-i	80386sx	xt/at	os2-1.1	1:2	2:74	3:248	4:235	559
86	ibm	80386	mca	o-1.1ee	1:1	2:69	3:244	4:234	548
70	ibm	8088	xt/at	dos3.2	1:0	2:65	3:240	4:233	538

The first seven test cases, those listed in PART 1, are sufficient to cover all 17 factor values, taken one factor at a time.

Continuing with the PART 2 cases will cover all *possible* combinations of factor values taken two at a time. There are 24 pairs of values which are not tested because of the constraints implied in the input test cases.

CATS OUTPUT -- PART 2									
8	ibm	80286	mca	dos4.0	1:0	2:61	3:236	4:232	529
13	compaq	80286	xt/at	dos3.3	1:0	2:57	3:232	4:231	520
31	at&t-o	80286	xt/at	dos3.2	1:0	2:54	3:228	4:230	512
39	at&t-i	80386	xt/at	dos3.3	1:0	2:51	3:224	4:229	504
98	at&t-o	80386	xt/at	os2-1.1	1:0	2:48	3:220	4:228	496
5	ibm	8086	xt/at	dos3.3	1:0	2:46	3:216	4:227	489
77	ibm	80286	mca	os2-1.1	1:0	2:43	3:213	4:226	482
15	compaq	80386	xt/at	dos3.1	1:0	2:41	3:210	4:225	476
18	compaq	80386	xt/at	dos4.0	1:0	2:39	3:207	4:224	470
25	at&t-o	8086	xt/at	dos3.1	1:0	2:37	3:204	4:223	464
38	at&t-i	80386sx	xt/at	dos4.0	1:0	2:35	3:201	4:222	458
90	ibm	80286	xt/at	o-1.1ee	1:0	2:33	3:198	4:221	452
22	ibm	80286	mca	dos3.2	1:0	2:32	3:195	4:220	447
72	ibm	8088	xt/at	dos4.0	1:0	2:31	3:192	4:219	442
91	compaq	80286	xt/at	os2-1.1	1:0	2:30	3:189	4:218	437
21	ibm	80286	mca	dos3.1	1:0	2:29	3:187	4:217	433
26	at&t-o	8086	xt/at	dos3.2	1:0	2:28	3:185	4:216	429
27	at&t-o	8086	xt/at	dos3.3	1:0	2:27	3:183	4:215	425
69	ibm	8088	xt/at	dos3.1	1:0	2:26	3:181	4:214	421
71	ibm	8088	xt/at	dos3.3	1:0	2:25	3:179	4:213	417
82	ibm	80386sx	mca	o-1.1ee	1:0	2:24	3:177	4:212	413

If the tester believes triple mode or quadruple mode failures are important, the test cases given in PARTS 3 and 4 can be considered.

CATS OUTPUT -- PART 3									
7	ibm	80286	mca	dos3.3	1:0	2:24	3:175	4:211	410
14	compaq	80286	xt/at	dos4.0	1:0	2:24	3:173	4:210	407
20	ibm	80386sx	mca	dos4.0	1:0	2:24	3:171	4:209	404
37	at&t-i	80386sx	xt/at	dos3.3	1:0	2:24	3:169	4:208	401
41	ibm	80386	mca	dos3.3	1:0	2:24	3:167	4:207	398
42	ibm	80386	mca	dos4.0	1:0	2:24	3:165	4:206	395
81	ibm	80386sx	mca	os2-1.1	1:0	2:24	3:163	4:205	392
85	ibm	80386	mca	os2-1.1	1:0	2:24	3:161	4:204	389
6	ibm	8086	xt/at	dos4.0	1:0	2:24	3:160	4:203	387
11	compaq	80286	xt/at	dos3.1	1:0	2:24	3:159	4:202	385
12	compaq	80286	xt/at	dos3.2	1:0	2:24	3:158	4:201	383
17	compaq	80386	xt/at	dos3.3	1:0	2:24	3:157	4:200	381
32	at&t-o	80286	xt/at	dos3.3	1:0	2:24	3:156	4:199	379
33	at&t-o	80286	xt/at	dos4.0	1:0	2:24	3:155	4:198	377
40	at&t-i	80386	xt/at	dos4.0	1:0	2:24	3:154	4:197	375
56	at&t-o	80386	xt/at	dos3.2	1:0	2:24	3:153	4:196	373
57	at&t-o	80386	xt/at	dos3.3	1:0	2:24	3:152	4:195	371
58	at&t-o	80386	xt/at	dos4.0	1:0	2:24	3:151	4:194	369
62	at&t-o	80286	xt/at	dos3.1	1:0	2:24	3:150	4:193	367
78	ibm	80286	mca	o-1.1ee	1:0	2:24	3:149	4:192	365
89	ibm	80286	xt/at	os2-1.1	1:0	2:24	3:148	4:191	363
92	compaq	80386	xt/at	os2-1.1	1:0	2:24	3:147	4:190	361
96	at&t-o	80286	xt/at	os2-1.1	1:0	2:24	3:146	4:189	359
101	at&t-i	80386	xt/at	os2-1.1	1:0	2:24	3:145	4:188	357

There are 145 three-at-a-time combinations and 185 four-at-a-time combinations which are untested because of the constraints.

CATS OUTPUT -- PART 4									
2	ibm	80286	xt/at	dos3.2	1:0	2:24	3:145	4:187	356
3	ibm	80286	xt/at	dos3.3	1:0	2:24	3:145	4:186	355
4	ibm	80286	xt/at	dos4.0	1:0	2:24	3:145	4:185	354

There is no further improvement in the number of untested combinations. All the rest of the test cases (PART 5) are redundant. These test cases need not be executed *if the factors under consideration are the only ones that matter.*

CATS OUTPUT -- PART 5									
9	ibm	80286	mca	dos3.3	1:0	2:24	3:145	4:185	354
10	ibm	80286	mca	dos4.0	1:0	2:24	3:145	4:185	354
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
97	at&t-o	80286	xt/at	os2-1.1	1:0	2:24	3:145	4:185	354
99	at&t-o	80386	xt/at	os2-1.1	1:0	2:24	3:145	4:185	354

3.4 Ease of Use

When there are dependencies among test factors, CATS does not require these dependencies to be recognized or described systematically. So the user need not analyze these dependencies. Instead, the

input must include all test cases to be considered. And sometimes there may be a large number of test cases. Preprocessing to generate CATS test cases has been done with shell routines, and with OATS.

3.5 Number of Suggested Test Cases

One of the properties of orthogonal arrays is that for any pair of columns, all combinations of factor values occur an equal number of times.[3] This property can be useful for associating a fault with a particular factor value. CATS makes no attempt to balance combinations of factor values. I.e. we opt to have testers and developers debug to locate the fault.[4] CATS attempts to minimize the number of untested combinations, so the objective is to cover combinations *at least once* rather than an equal number of times. Sometimes, to meet this simpler objective, only a subset of the test cases given by an orthogonal array is needed. For example, in the L64 orthogonal array every pair of columns contains the four combinations of factor values (1 1, 1 2, 2 1, and 2 2) sixteen times. Only 16 rows (test cases) of this array are needed to cover each of the combinations of factor values at least once. The following table shows the L64 array with rows reordered to cover the combinations at least once, but not an equal number of times.

3.6 Order of Test Case Execution

These examples demonstrate that CATS not only suggests cases to test: A preferable order of tests is also given. Executing the tests in the suggested order gives early coverage to the most combinations of factor values. The intent is to find problems sooner -- It is desirable to find faults early in the test phase of a project because there is more information for managing the bug fixing process, and because it takes time to fix the bugs.

4. Description of CATS

CATS is a single program written in C language. It uses two arrays to store and manipulate test cases -- an *experiment* array containing test cases we assume have been performed, and a *vector* array containing all the remaining test cases, any one of which we could do next. Initially the experiment array is empty, and all the test cases are placed in the vector array.

On a trial basis, each test case from the vector array is placed in the experiment array, and the number of untested combinations in the experiment array is found. The best test case -- the one with the fewest untested combinations -- is selected. It is placed in the experiment array, to be done next; and it is removed from the vector array, as it no longer indicates a new direction.

The process of finding the next best test case is repeated, and each corresponding test case is moved from the vector array to the experiment array. This goes on until all the test cases are moved from the vector array to the experiment array, or there is no further improvement in the test coverage.

Essentially, CATS follows the steepest gradient through the space of test cases, to reach the minimum number of untested combinations of factor values.

CATS OUTPUT – L64 ORTHOGONAL ARRAY

-2	222222222222222222222222222222222222222222222222222222222222222	2:7812	7812	63:9223372036854775800
1	111111111111111111111111111111111111111111111111111111111111111	2:5859	5859	
2	111111111111111111111111111111122222222222222222222222222222222	2:4371	4371	
3	111111111111111222222222222222211111111111111112222222222222222	2:3243	3243	
5	111111122222222111111112222222211111111222222221111111122222222	2:2391	2391	
9	111222211112222111122221111222211112222111122221111222211112222	2:1749	1749	
31	122221122111122221111221122111111222211221111222211112211222211	2:1254	1254	
16	111222222221111222211111111222222221111111122221111222222221111	2:886	886	
33	212121212121212121212121212121212121212121212121212121212121212	2:618	618	
49	221122112211221122112211221122112211221122112211221122112211221	2:419	419	
20	122112211221122221122112211221122112211221111221122112211221122	2:283	283	
37	212121221212121121212122121212112121212212121211212121221212121	2:191	191	
58	221211212212112122121121221211221121221211212212112122121121221	2:123	123	
4	111111111111111222222222222222222222222222222221111111111111111	2:75	75	
43	212212112122121212112122121121212122121121221212121121221211212	2:41	41	
64	221211221121221211212211221211221121221122121121221211221121221	2:16	16	
23	122112222112211221122111122112211221122222211221122112211112211 22	2:0	0	
6	111111122222222111111112222222222222222111111112222222211111111	2:0	0	
7	111111122222222222222221111111111111111222222222222222211111111	2:0	0	
8	111111122222222222222221111111122222222111111111111111122222222	2:0	0	
10	111222211112222111122221111222222221111222211112222111122221111	2:0	0	
11	111222211112222222211112222111111112222111122222222111122221111	2:0	0	
12	111222211112222222211112222111122221111222211111111222211112222	2:0	0	
13	111222222221111111122222222111111112222222211111111222222221111	2:0	0	
14	111222222221111111122222222111122221111111122222222111111112222	2:0	0	
15	111222222221111222211111111222211112222222211112222111111112222	2:0	0	
17	122112211221122112211221122112211221122112211221122112211221122	2:0	0	
18	122112211221122112211221122112222112211221122112211221122112211	2:0	0	
19	122112211221122221122112211221111221122112211222211221122112211	2:0	0	
21	122112222112211112211222211221111221122221122111122112222112211	2:0	0	
22	122112222112211112211222211221122112211112211222211221111221122	2:0	0	
24	122112222112211221122111122112222112211112211221122112222112211	2:0	0	
25	122221111222211112222111122221111222211112222111122221111222211	2:0	0	
26	122221111222211112222111122221122111122221111222211112222111122	2:0	0	
27	122221111222211221111222211112211222211112222112211112222111122	2:0	0	
28	122221111222211221111222211112222111122221111221122221111222211	2:0	0	
29	122221122111122112222112211112211222211221111221122221122111122	2:0	0	
30	122221122111122112222112211112222111122112222112211112211222211	2:0	0	
32	122221122111122221111221122221122111122112222111122221122111122	2:0	0	
34	212121212121212121212121212121221212121212121212121212121212121	2:0	0	
35	212121212121212212121212121212112121212121212122121212121212121	2:0	0	
36	212121212121212212121212121212121212121212121211212121212121212	2:0	0	
38	212121221212121121212122121212121212121121212122121212112121212	2:0	0	
39	212121221212121212121212121212121212121212121221212121211212121 2	2:0	0	
40	212121221212121212121211212121221212121121212121212121221212121	2:0	0	
41	212212112122121121221211212212112122121121221211212212112122121	2:0	0	
42	212212112122121121221211212212121211212212112122121121221211212	2:0	0	
44	212212112122121212112122121121221211212212112121212212112122121	2:0	0	
45	212212121211212121221212121121212212121211212121221212121211212	2:0	0	
46	212212121211212121221212121121221211212121221212121121212122121	2:0	0	
47	212212121211212212112121212212112122121212112122121121212122121	2:0	0	
48	212212121211212212112121212212121211212121221211212212121211212	2:0	0	
50	221122112211221122112211221122121122112211221122112211221122112	2:0	0	
51	221122112211221211221122112211212211221122112212112211221122112	2:0	0	
52	221122112211221211221122112211221122112211221121221122112211221	2:0	0	
53	221122121122112122112212112211212211221211221121221122121122112	2:0	0	
54	221122121122112122112212112211212211221122112212112211212211221	2:0	0	
55	221122121122112211221121221122112211221211221121122112122112 21	2:0	0	
56	221122121122112211221121221122112211221211221121221122121122112	2:0	0	
57	221211212212112122121121221211212212112122121121221211212212112	2:0	0	
59	221211212212112112212122121121212212112121221211221121221211221	2:0	0	
60	221211212212112112212112122121121221211212212112212211212212112	2:0	0	
61	221211221121221122121122112122112212112211212211221211221121221	2:0	0	
62	221211221121221122121122112121221122122121121122121221122122112	2:0	0	
63	221211221121221211212212211212211221211212212211221211221212112	2:0	0	

5. Summary

The Constrained Array Test System is a new tool which can help to save test time and improve the coverage of test factors. It conforms to constraints among test case factors simply, and makes test case recommendations to minimize the number of untested combinations of factor values. The combinations can be based on an arbitrary number of factors, to locate single mode, double mode or higher mode faults.

CATS suggests a preferable order to perform the tests, so that combinations of factor values are covered as early as possible in the test process.

The CATS prototype is available without formal support to interested users.

The author thanks Bob Brownlie and Madhav Phadke for stimulating and helpful discussions on this topic.

[signed]

G. B. Sherwood

Att.
REFERENCES
CATS Manual Pages

Copy to
XT9152400 Supervision
Workgroup Computing Integration and Test Group
M. D. Balkovic
F. W. Bennett
L. Bernstein
R. B. Brandt
R. A. Brownlie
A. P. Chintapalli
G. C. Darden
D. J. Dooling
C. J. Harkness
S. S. Hegde
J. R. Kane
K. J. Kennedy
D. J. Morgan
M. S. Phadke
E. H. Stredde
S. A. Tartarone

REFERENCES

1. A. P. Chintapalli, S. S. Hegde, M. S. Phadke, "Increase Software/Hardware Test Effectiveness with OATS," September 21, 1989.

2. D. R. Antolick, M. A. Dennis, L. J. Schroeder, "StarGROUP/StarLAN Release 3.3 Configuration Requirements, Issue 2.0," September 19, 1989. (See also M. C. Streff for subsequent MRs.)

3. M. S. Phadke, "Quality Engineering Using Robust Design" (Chapter 3), 1989, Prentice Hall.

4. See also R. C. Eisele, R. L. Vanderwall, "A Comparison of Cause-Effect Matrices and Orthogonal Matrices in Software Testing," October 31, 1989.

NAME

cats – Constrained Array Testing System

SYNOPSIS

cats

DESCRIPTION

cats reads an array of values of factors representing test cases from the standard input. Then it reorders the cases to minimize the number of untested combinations of factor values. The test cases are written to the standard output with a running count of untested combinations appended to each case.

Combinations of factor values can be analyzed three ways: The number of factors to consider at a time can be specified by starting the "zeroth" row of input data with the negative of the number of factors to consider. For example, for *pairs* of factors the zeroth row starts with -2. The *maximum* number of factors to consider at a time can be specified by starting the zeroth row of data with the maximum number of factors to consider. For example, for factor values to be considered one-at-a-time and two-at-a-time and three-at-a-time, the zeroth row starts with 3. By default *cats* considers as many factors as possible, limited by the number of factors in the data or by computing resources. In this case, the zeroth row starts with 0.

EXTERNAL INTERFACES

Input to this tool follows the format used for orthogonal arrays in the *oats*(1T) data subdirectory. Each row of data starts with its row number, 0, 1, 2, etc. The zeroth row contains the number of values or levels for each factor, separated by white space. Each subsequent row contains character strings corresponding to the values for the factors.

On output the values are separated by tab characters. Appended to each row are the numbers of untested combinations of factor values for the indicated number of factors at a time. The total number of untested factor values is also given. The zeroth row gives the number of untested combinations before any tests are performed; each subsequent row contains the numbers of remaining untested combinations, assuming test cases are executed in the indicated order.

When the number of factors in the problem exceeds the maximum number under consideration, the zeroth row also indicates the number of factor values when all factors are considered. This is the maximum number of test cases without any constraints.

EXAMPLE

SAMPLE INPUT:

0	2	3	2
1	one	red	value_1
2	two	red	value_2
3	one	green	value_1
4	one	green	value_2
5	one	blue	value_2
6	one	blue	value_2
7	one	red	value_1
8	two	red	value_2
9	one	green	value_1
10	one	green	value_2
11	two	blue	value_2
12	two	blue	value_2

There are 3 factors (columns), each of which can have 2 or 3 values. 12 test cases (rows) are listed.

SAMPLE OUTPUT:

0	2	3	2	1:7	2:16	3:12	35
1	one	red	value_1	1:4	2:13	3:11	28
11	two	blue	value_2	1:1	2:10	3:10	21
4	one	green	value_2	1:0	2:7	3:9	16
2	two	red	value_2	1:0	2:5	3:8	13
3	one	green	value_1	1:0	2:4	3:7	11
5	one	blue	value_2	1:0	2:3	3:6	9
6	one	blue	value_2	1:0	2:3	3:6	9
7	one	red	value_1	1:0	2:3	3:6	9
8	two	red	value_2	1:0	2:3	3:6	9
9	one	green	value_1	1:0	2:3	3:6	9
10	one	green	value_2	1:0	2:3	3:6	9
12	two	blue	value_2	1:0	2:3	3:6	9

There are $2 + 3 + 2 = 7$ combinations of values for individual factors. There are $2 * 3 + 2 * 2 + 3 * 2 = 16$ combinations of values for pairs of factors. There are $2 * 3 * 2 = 12$ combinations of values for factors taken three at a time.

Three test cases (1, 11 and 4) are needed to cover all individual values. The first six test cases are sufficient to cover all possible pairs of factor values. The remaining six cases are redundant. A total of nine combinations are not covered due to the constraints of the input data.

SEE ALSO

oats(1T).

WARNINGS

Row one must start with **1**. Other rows can be identified with arbitrary strings without white space.

BUGS

Command line arguments and comments in the input data are not supported.